Traffic Analysis (what, how, and how)

TDTS21 Advanced Networking

Ethan Witwer, April 2025

LINKÖPING UNIVERSITY

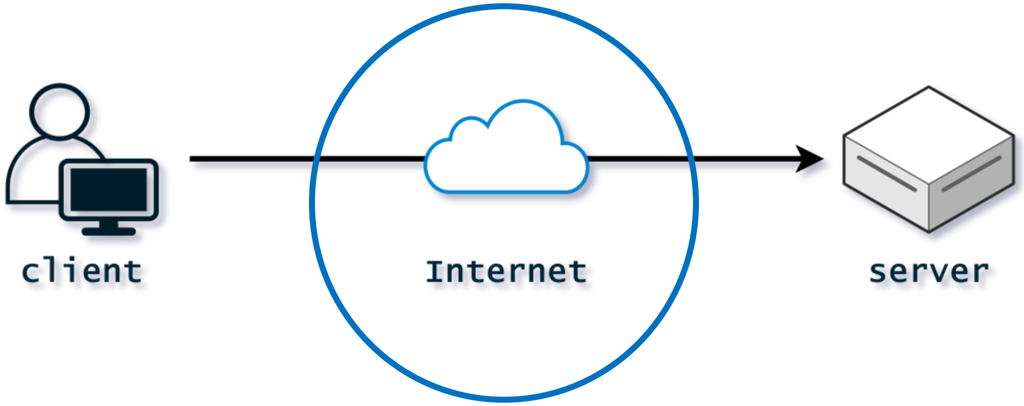
Part 1

Background, threat model, and attacks

Client-Server Model, Simplified

User perspective:

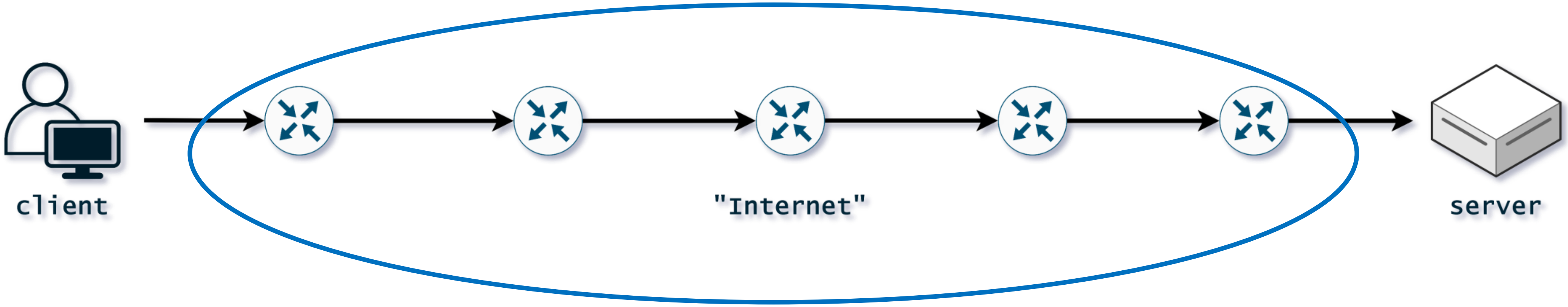
- send a *request* to some server
- receive a *response* with some data



Client-Server Model, More Detailed

Network perspective:

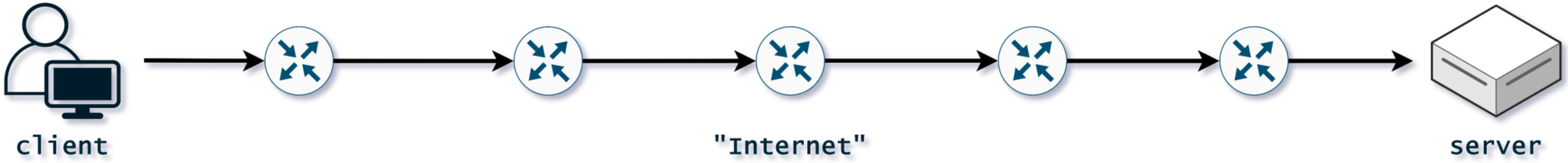
- send a *request* to some server
- … through the local network, your ISP's network, the Internet backbone, the server's ISP's network, …



Client-Server Model, Implications

Many actors can see your traffic, what now?

- Use encryption to protect sensitive data (HTTPS)
- Connect to a VPN if the local network is untrusted



Client-Server Model, Implications

Is that sufficient?

(Apart from the fact that it’s up to the webmaster to set up HTTPS, and you need to trust your VPN provider…)

The answer is not a definite yes.

Traffic Analysis: Bypass Encryption

Analyze patterns in encrypted traffic

1. *packet size* – MTU? smaller? variable?
2. *packet timing* – when are packets sent in relation to each other? interval between packets?
3. *packet direction* – are packets being sent to the server or received by the client?

Case Study 1: File Download

User's goal:

1. send a request to an FTP server
2. receive the file contents in one bulk download
3. use encryption so that file contents are not exposed to network observers

Case Study 1: File Download

Attacker's goal:

1. analyze the traffic, learn which file is being downloaded by the user
2. this must be done without breaking encryption

How can this be done? Technique #1 - packet sizes

Case Study 1: File Download

Attacker's algorithm:

1. **ground truth,** connect to the server and gather a mapping between file names and sizes (lsdir)
2. **observe traffic,** sum the sizes of packets sent from server to client to approximate file size
3. **classification,** match with directory listing

Case Study 1: File Download

Two of the assumptions implicit here:

- Importantly, *ground truth can be obtained*.
  - The attacker may not have access to the server, not know where the server is (the user is connected to a VPN/Tor), not have permission to list files, …
- Also, *the encryption algorithm must output ciphertext similar to the plaintext in length*.
  - The encrypted file must be about the same size as (or a predictable function of) the original

Fingerprinting Attacks (Classification)

- A canonical type of traffic analysis attack
- Some features from the encrypted traffic are matched against ”fingerprints” of known resources
- Thus, these features (or some) must be present every time a particular resource is accessed
- It must also be possible to generate fingerprints: consider the file download example

Fingerprinting Attacks (Classification)

The three stages described for file downloads can be generalized to typical fingerprinting attacks:

1. **ground truth:** generate fingerprints
2. **observe traffic:** save details about packets, potential feature engineering / transformations
3. **classification:** some method to match fingerprints with (features of) encrypted traffic

Case Study 2: Web Page Download

User's goal:

1. send a request to an HTTP server
2. receive the web page's main file (likely HTML)
3. iteratively request embedded external resources
4. use encryption so that resource contents are not exposed to network observers

Case Study 2: Web Page Download

Attacker's goal:

1. analyze the traffic, learn which web page (or, more broadly, website) is being downloaded by the user
2. this must be done without breaking encryption

How can this be done? No longer so clear…

IP address, SNI, etc. cannot be used (VPN / Tor)

Case Study 2: Web Page Download

Web traffic is more of a black box:

- web pages can consist of **many resources**
- browser behavior, resource downloads **overlap**
- no obvious, intuitive way to identify websites

Case Study 2: Web Page Download

World size is a significant factor:

- Even if we figure out which features are useful, can we generate a fingerprint for all websites?
- Would we have the time and computational power to match against that many fingerprints?
- Different web pages with the same content, some have multiple versions (localization), updates to content, ...

Web Page Download, early research

Heuristics, hand-crafted features, and small worlds

A relatively early example attack (2016): CUMUL[1]

$$T = (p_1, \ldots, p_N)$$

$$C(T) = ((0,0), (a_1, c_1), \ldots, (a_N, c_N))$$

$$c_1 = p_1 \qquad c_i = c_{i-1} + p_i$$

$$a_1 = |p_1| \qquad a_i = a_{i-1} + |p_i|$$

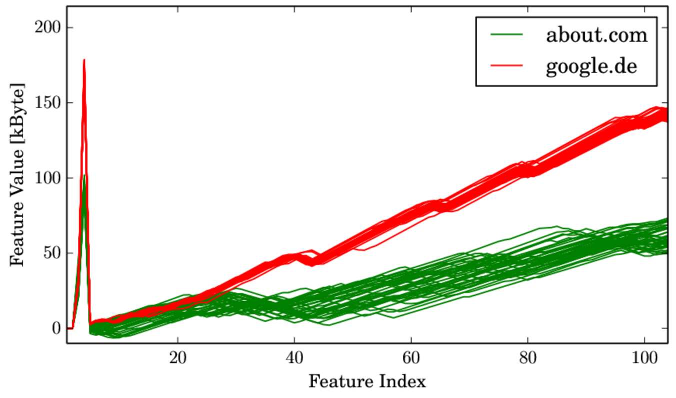
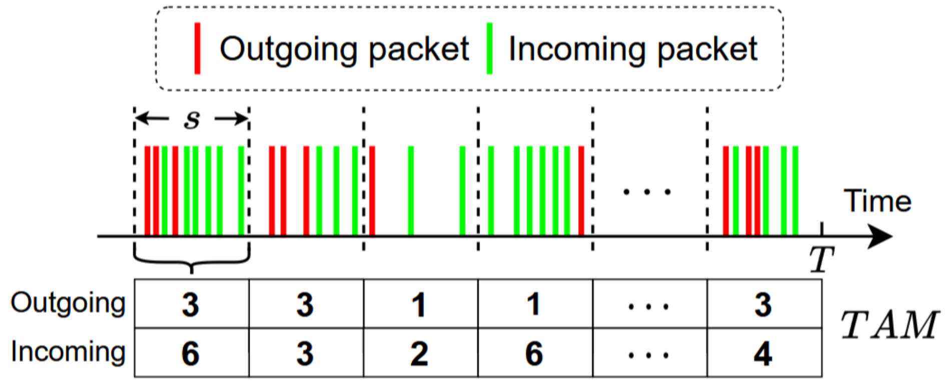


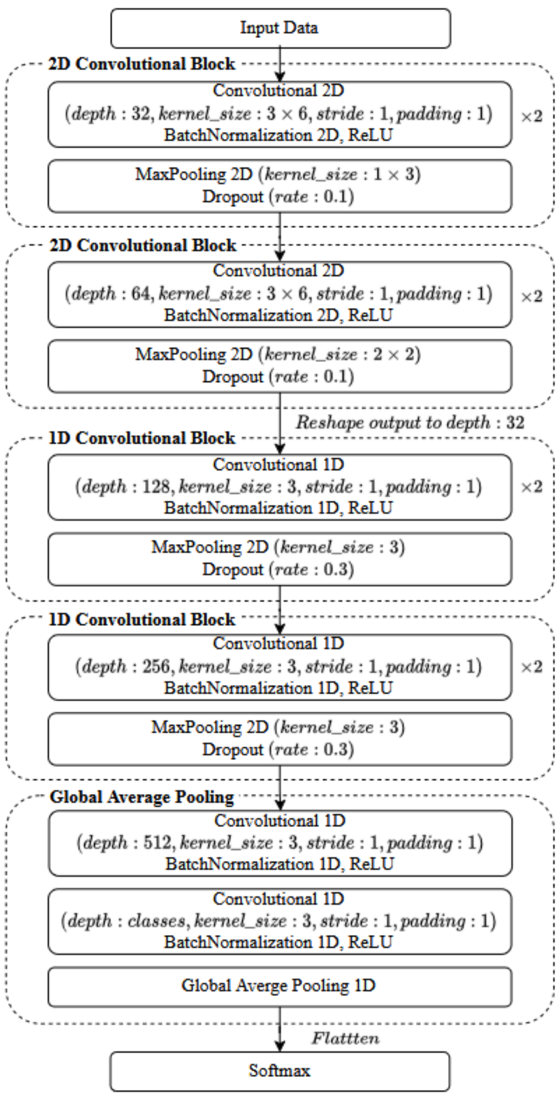
Fig. 2: Visualized fingerprints of two websites

Web Page Download, recent research

Deep learning with automatic feature extraction

State-of-the-art attack: Robust Fingerprinting[2]





Web Page Download, evaluation

How do the three stages come in?

- **ground truth:** collect fingerprints for *some* websites, potentially implicitly stored in a model (gather feature representations and train with them)
- **observe traffic:** generate the same feature representations for observed web page visits
- **classification:** test the model

Web Page Download, evaluation

- **ground truth:** collect fingerprints for *some* websites, potentially implicitly stored in a model (gather feature representations and train with them)

Which websites? When and how to collect data?

Also a can of worms…

Web Page Download, evaluation

In research:

- closed- vs. open-world evaluation
- popular websites most often used
  - Alexa[3]
  - Open PageRank Initiative[4]
- homepages and subpages[3,4]
- genuine measurements from Tor exit nodes[5]

Web Page Download, evaluation

In reality, unclear. Some questions:

- which websites may be visited by users?
- which classifier is being used, and how does it behave when fed different types of data?
- which network conditions do users have? how do these change over time?
- where is the attack to be performed?

Part 2

Defenses, frameworks, and more

Case Study 1: File Download

User's *updated* goal:

1. send a request to an FTP server
2. receive the file contents in one bulk download
3. use encryption so that file contents are not exposed to network observers
4. have some defense to prevent traffic analysis from exposing the file

Case Study 1: File Download

A simple defense:

- locate the largest file on the server, with size X
- send extra data from server to client with every download so that X bytes are always downloaded, no matter which file is requested

```
-rw-r--r--  1 root  root  449848 apr  7 13:09 s81.png
-rw-r--r--  1 root  root  544821 apr  7 13:09 s82.png
-rw-r--r--  1 root  root  361499 apr  7 13:09 s83.png
-rw-r--r--  1 root  root  181375 apr  7 13:09 s84.png
-rw-r--r--  1 root  root  215032 apr  7 13:09 s85.png
-rw-r--r--  1 root  root  215502 apr  7 13:09 s86.png
-rw-r--r--  1 root  root   91647 apr  7 13:09 s87.png
```

What are the results?

- perfect **protection** against file fingerprinting
- high **overhead:** what if the biggest file is 5 GB larger than most other files on the server?

Case Study 1: File Download

A generalization:

- group files with similar sizes together
- send extra data from server to client with every download so that all files in group X appear to have some size Y (size of largest file in group)

```
-rw-r--r-- 1 root root 245301 apr  7 09:56 099.1000.mdd.json
-rw-r--r-- 1 root root 248799 apr  7 09:56 099.2000.mdd.json
-rw-r--r-- 1 root root 233450 apr  7 09:56 099.4000.mdd.json
-rw-r--r-- 1 root root  40733 apr  7 09:56 099.audio.mdd.json
-rw-r--r-- 1 root root 245301 apr  7 09:56 100.1000.mdd.json
-rw-r--r-- 1 root root 248799 apr  7 09:56 100.2000.mdd.json
-rw-r--r-- 1 root root 233450 apr  7 09:56 100.4000.mdd.json
-rw-r--r-- 1 root root  40733 apr  7 09:56 100.audio.mdd.json
```

What are the results?

- **tunable** defense against file fingerprinting
- **trade-off** between protection and overhead

Theory and Practice

These defenses provide theoretical guarantees…

…as long as no other influencing factors are present

- Requests from client to server
- Control/status messages in download protocol
- Different response delay depending on file

Case Study 2: Web Page Download

User's goal:

1. send a request to an HTTP server
2. receive the web page's main file (likely HTML)
3. iteratively request embedded external resources
4. use encryption so that resource contents are not exposed to network observers

Case Study 2: Web Page Download

Web traffic is more of a black box:

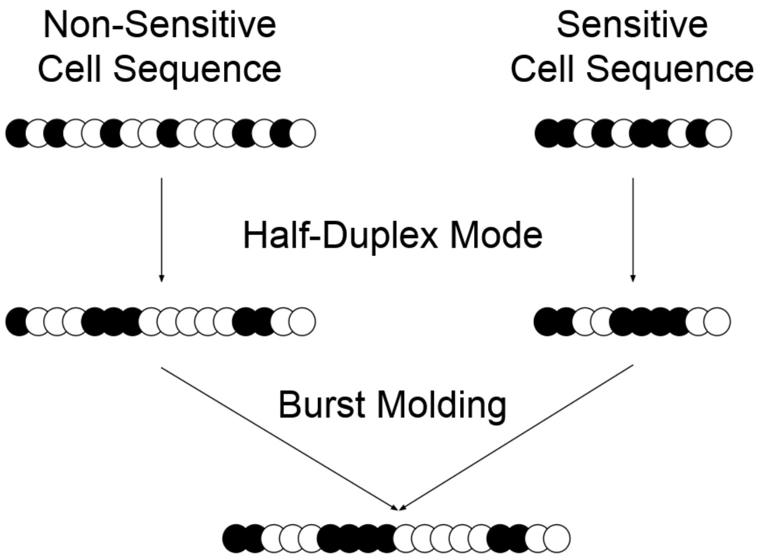
- web pages can consist of **many resources**
- browser behavior, resource downloads **overlap**
- no obvious, intuitive way to identify websites

Given this, how can we <u>defend</u> web traffic?

A Theoretically Backed Defense: Walkie-Talkie

Browser in half-duplex mode, proxy cooperation[6]

”Burst molding” to create *explicit anonymity sets*



☑ size

☑ direction

timing?

A Theoretically Backed Defense: Walkie-Talkie

Tik-Tok[7]:

- 49.7% accuracy
- 98.4% top-2 accuracy

The cost?

- 31% *bandwidth*
- 34% *latency*

Trade-offs: A Closer Look

Desirable features of a defense

- **high protection:** mitigate fingerprinting attacks
- **low latency overhead:** retain user experience, packet delays can lead to slower web page loads
- **low bandwidth usage:** save network capacity, and avoid indirect effects on user experience[8]

The Problem with Theory

Desirable features of a defense

- **high protection:** mitigate fingerprinting attacks

Can this be guaranteed with low overheads?

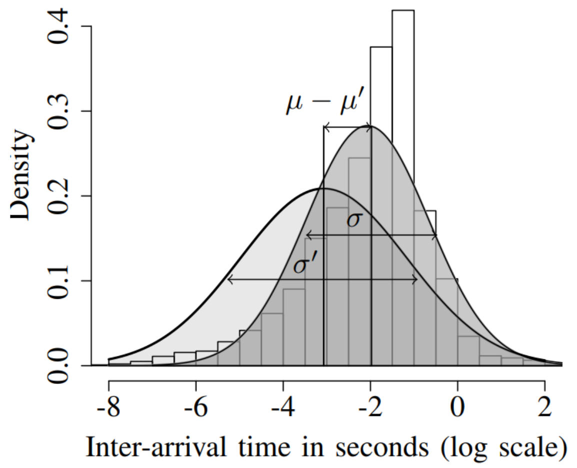
And minimal to no impact on user experience?

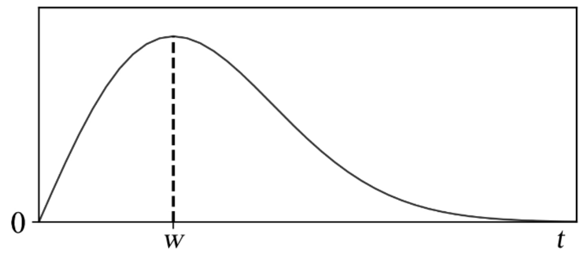
Research says: ”Strong Anonymity, Low Bandwidth Overhead, Low Latency—Choose Two”[11]

Padding-Only Defenses

Many defenses avoid delays entirely

- **WTF-PAD**[9]: first candidate for Tor
  - burst and gap mode
  - defeated, nearly useless
- **FRONT**[10]: based on observations about attacks
  - two parameters, Rayleigh distribution
  - defeated, nearly useless





Padding-Only Defenses

Observations from padding defenses:

- Attacks and defenses are an arms race
- Hard-coded defenses are thus undesirable
- Padding is often randomized

Tor Circuit Padding Framework

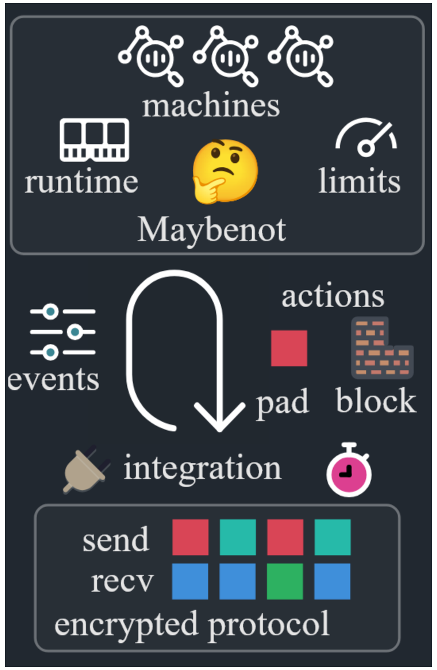
Implement *building blocks* for defenses[12]

- non-deterministic finite state machines
- event-driven framework, only padding actions
- histograms/distributions for inter-packet times

Maybenot Framework

Improve upon the circuit padding framework[13]

- probabilistic finite state machines
- many events, padding and blocking actions
- no histograms, distributions sampled often
- standalone library

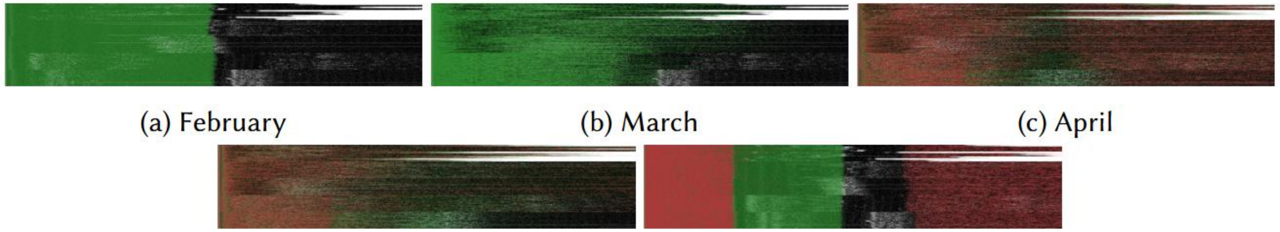


Defense Generation

Genetic programming: Tor circuit padding[3]

sent non-padding, **sent padding**

received non-padding, **received padding**



(a) February (b) March (c) April

(d) May (e) June

Traffic Analysis (what, how, and how)

TDTS21 Advanced Networking

Ethan Witwer, April 2025

LINKÖPING UNIVERSITY

References

1. Panchenko et al. Website Fingerprinting at Internet Scale

2. Shen et al. Subverting Website Fingerprinting Defenses with Robust Traffic Representation

3. Pulls. Towards Effective and Efficient Padding Machines for Tor

4. Mathews et al. SoK: A Critical Evaluation of Efficient Website Fingerprinting Defenses

5. Jansen et al. A Measurement of Genuine Tor Traces for Realistic Website Fingerprinting

6. Wang et al. Walkie-Talkie: An Efficient Defense Against Passive Website Fingerprinting Attacks

7. Rahman et al. Tik-Tok: The Utility of Packet Timing in Website Fingerprinting Attacks

8. Witwer et al. Padding-only Defenses Add Delay in Tor

9. Juarez et al. WTF-PAD: Toward an Efficient Website Fingerprinting Defense for Tor

10. Gong et al. Zero-delay Lightweight Defenses against Website Fingerprinting

11. Das et al. Anonymity Trilemma: Strong Anonymity, Low Bandwidth Overhead, Low Latency—Choose Two

12. Tor Project. Tor's Circuit Padding Framework

13. Pulls et al. Maybenot: A Framework for Traffic Analysis Defenses